UML Implementation of Music Recommender System

V.S. Sai Supreetha
4/4 Btech (Information Technology)
Bapatla Engineering College
Bapatla, India
supreetha@outlook.com

V.S.S.K. Anand
4/4 Btech (Information Technology)
Bapatla Engineering College
Bapatla, India
anand_fun30@yahoo.com

Abstract
There is large amount of music digitally available on the internet. Music recommender simplifies the task of finding out what music a user might like and provide recommendations. The aim of this paper is to give unified modelling language implementation of a music recommender system. As there is no standard way of representing a recommender system structurally because of the number of ways they can be constructed for varied platforms and inherent complex nature of the systems themselves, this paper tries to define a highly abstractive UML implementation[5] of a collaborative filtering model[1] for recommending music mainly focussing on the usecase, activity, class and sequence diagram. The defined UML designs are such that they can be easily adapted to any other recommender system with very little behavioral change.

Keywords: Recommender System, UML, Collaborative Filtering.

1. INTRODUCTION:

Recommender systems are software agents that elicit the interests and preferences of individual consumers and make recommendations accordingly[8]. These are basically the systems that recommend things like music, videos, books, shopping items, and even people. They have the potential to support and improve the quality of the decisions consumers make while searching for and selecting things online. They have become overly popular in the recent times with their presence and increase in their use on almost every platform. The most popular live recommender systems can be seen on platforms like Amazon, Facebook, Youtube, Last.fm etc. Their need has been largely increased because of the very size of the population to which these platforms cater to. They basically improve user experience. A user, for example, would not like to go through the hassle of finding something in the very big inventory of, say, Amazon and would highly appreciate if an item is being recommended to him based on some criteria like his rating of previously bought things or his most favourite category etc.

The music recommender systems are double edged swords. The are of valuable use both to the user as well as the provider. They keep the user engaged by finding interesting music in the form of recommendations, lessening the burden on the user by reducing the set of choices to choose from. They give the scope for exploration and discovery of music that the user may not know exists. Because it is a music recommender there is never less entertainment.

Providers can benefit by giving personalized service to each user which increases user loyalty thereby increasing user activity on their site. They can construct knowledge base models and systems from large amount of data gathered, make money by promotion and persuasion[8].

Music streaming sites like Last.fm and Spotify work in this fashion in that they recommend music and in some cases forms a playlist of songs automatically. The important thing to note here is that they are dynamic and keep changing the recommendations with the user’s activity. They use machine learning algorithms to keep track of things so they evolve with every user rating of the recommendations.

Recommender systems can be built in many ways:

- Personalized: Here user’s profile and his/her context is only considered.
- Collaborative: Here all users data is considered and recommendations are constructed.
- Content-based: Here similar items of the those positively rated by user are recommended.
- Knowledge-based: Here both user context and product attributes are considered.

© IJMSET-Advanced Scientific Research Forum (ASRF), All Rights Reserved
"IJMSET promotes research nature, Research nature enriches the world’s future"
Hybrid: Here different compositions of all the above methods are used.

In this paper a UML design to a highly abstract music recommender system is implemented. Unified modelling languages is a standardized language used for design specifications in software engineering. As recommender systems are machine learning systems which do not follow a specific methodology and are constructed according to the need and the context of application, and their modelling is done in an ad-hoc way, a UML implementation is given here[4]. Models generally help in depicting every singular feature of the system for better understanding. Moreover, UML has the object oriented features best suitable for the current scenario. Using this model any music recommender system can be designed with any algorithm as this is abstracting the core functionality but defining the high level functional elements.

For developing this music recommender system an MVC architecture is followed. The ‘model’ holds the data sources consisting of data regarding the music, artists as well as the users and their activity. The ‘view’ is the user interface to the system, the various elements and their involvement in the recommendation algorithm. The ‘controller’ is the middle layer which consists of the business logic for the recommender system.

2. METHODOLOGY:

The problem statement is quite straightforward: recommend music to users from a large repository of songs. There are many live online music recommender systems each following its own methodology in recommending music. The Last.fm, for example follows a methodology where it takes into consideration the user activity, the similarity scores based on tags, artists and the tracks. The Pandora music service works on the basis of the surveys done by the musicologists. Here we are taking a very basic approach of calculating similarities of artists and songs, based on listening events of the users.

The artists and songs are represented as vectors of the playcount by each user: 

\[ A_i = (c_1, c_2, \ldots, c_n) \text{ and } S_j = (p_1, p_2, \ldots, p_n) \]

where \( A_i \) represents \( i \) th artist and \( i \) is from 1 to \( m \) (i.e., there are \( m \) artists), \( S_j \) represents \( j \) th song and \( j \) is from 1 to \( r \) (i.e., there are \( r \) songs); \( c_1, c_2, \ldots, c_n \) are the number of times the users \( u_1, u_2, \ldots, u_n \) listened to an artist and \( p_1, p_2, \ldots, p_n \) represents number of times the users \( u_1, u_2, \ldots, u_n \) listened to a song. Now the algorithm to be applied is just a Pearson correlation\[6\] measure between the vectors of corresponding type:

\[
\text{sim}(a, b) = \frac{\sum_{p \in P}(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b)}{\sqrt{\sum_{p \in P}(r_{a,p} - \bar{r}_a)^2} \sqrt{\sum_{p \in P}(r_{b,p} - \bar{r}_b)^2}}
\]

Where,

\( a, b \) are the two vectors of same type,

\( 'p' \) is a user belonging to set of users \( P \),

\( r_{a,p} \) is the rating for song/artist represented by vector ‘\( a \)’; similarly \( r_{b,p} \)

\( \bar{r}_a \) represents the average rating by all users for the song/artist represented by vector ‘\( a \)’; similarly \( \bar{r}_b \).

Similarity score for artists/songs:

This helps to generate a list of similar songs/artists to the users request. Here no context i.e., the features of the music is considered and the list is basically full of suggestions of the songs/artists similar to the ones the user is currently listening to or had listened to. Now the prediction score which represents the likeliness of a song or an artist being recommended to the user from the list generated is given by:
\[ \text{pred}(a, p) = \bar{r}_a + \frac{\sum_{b \in N} \text{sim}(a, b) \ast (r_{b,p} - \bar{r}_b)}{\sum_{b \in N} \text{sim}(a, b)} \]

This is the weighted average of the similarity scores combined with the rating for the particular song/artist.

**UML Implementation of the System:**

Before going for the designing the structure and behavior of the system, in software engineering, requirements are gathered and analysed. Here the requirement analysing involves the context and scenarios where the system is going to be used. The user scenarios are used for constructing the Use case[7] diagrams. They basically depict the specific functionalities provided by the system to its users.

**Use case diagram:**

Here the actors are the user and the admin. Each use case comes with the description, actors, pre/post conditions and the flow of control. Here the admin is the one that maintains the application.

<table>
<thead>
<tr>
<th>Actors</th>
<th>Use cases</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>Search Music</td>
<td>Querying the system</td>
</tr>
<tr>
<td></td>
<td>Rate Music, Recommendations</td>
<td>Convey the liking of the music</td>
</tr>
<tr>
<td></td>
<td>Listen to Music</td>
<td>Play/Stop music</td>
</tr>
<tr>
<td></td>
<td>Access Recommendations</td>
<td>Get the recommended songs/artists</td>
</tr>
<tr>
<td>Admin</td>
<td>Gather User Data</td>
<td>Log user activities, Listen count etc.</td>
</tr>
<tr>
<td></td>
<td>Calculate Similarity Scores</td>
<td>Apply the pearson correlation</td>
</tr>
<tr>
<td></td>
<td>Show Predictions</td>
<td>Recommend from prediction scores</td>
</tr>
</tbody>
</table>

Coming to the conditions on the use cases, the implicit one is that the user must be logged into the system to access recommendations. Flow of control would be the inherent activities of each use case like the processing of the request as soon as the user enters a query for searching the music, the compiling of the list generated from the scores and showing those with high values etc.

**Activity Diagram:**

© IJMSET-Advanced Scientific Research Forum (ASRF), All Rights Reserved

“IJMSET promotes research nature, Research nature enriches the world’s future”
Main activities for each of the service provided are described by an activity diagram. This activity diagrams deal with the workflows of the recommender system. The work flows are depicted in sequence and often have conditions specified on the control-flow lines. The diagram is accompanied by the description, the initiator of the activity and the workflow. The initiator is generally a function module that gets called when an activity starts. Some of the functions in this music recommender system implementation are:

- `initMusicRecSys(…)` – create new instance of the system
- `loadUserProfile(…)` – authenticates and loads a user profile
- `processRequest(…)` – analyses the query entered by the user
- `dispResults()` – get the results and display them
- `logUserActivity(…)` – playcount, timestamp etc are all logged for further calculation
- `getRecommendation(…)` – show the recommendations after calculating similarity and prediction scores
- `logExplicitRating(…)` – user rating to specific song is noted for further calculations

Note that these are all parameterized methods and will be known during the implementation stage and are being omitted here.

![Activity diagram for Music Recommender System](image)

**Class Diagram:**

The class diagrams describe the structure of the system being modelled. They are the building blocks so to speak for object oriented modelling as with them comes all the object oriented concepts that exist among various individual components of the system. The three compartment figure of classes holds the name, the list of attributes and the operations.

In the current music recommender system being modelled, there are three main classes: the User, the Artist and the Song. The additional two classes Similarity and Recs are for calculating similarity scores and getting recommendations of songs and artists. These classes are just depicting the business logic part of the system and not the user interface. The Fig3 depicts the multiplicity also which tells the number of instances of one class referenced by the other in an association relationship:

- Each user is entitled to 0 or more recommendations of songs or artists.
- Each recommendation is based on the rating of the song by user and calculating similarity score for other with this song.
For Each song there may be 0 or more number of ratings.
Also, for each song there will be 0 or more number of similar songs.
For each artist there will be 0 or more similar artists.

**Sequence Diagram:**

The sequence diagrams are one of the interaction diagrams that depict the communication between the objects. The collaboration of objects is modelled based on a time sequence. The objects involved in the scenario pass messages between themselves. Here the return messages are not shown but are to be understood as implicit. The diagram below is basically asynchronous way of communication as in the recommender system the similarity scores are to be computed asynchronously without the user having to wait for long.
The RecSysUI object comes into picture here as it is the one with which the user communicates. The RecSysUI class has many visual elements that contribute to the data source in terms of user activity. The search button, the rating elements, the recommendation display section, the song player and play/stop buttons etc., all contribute to the logging of user activity. Since only behavioral nature of the recommender system is being discussed in the sequence diagrams, the user interface elements are abstracted out into a single class.

3. RESULTS AND DISCUSSION:

The purpose here is to just model the music recommender system using three behavioral and one structural diagram such that it depicts a wide range of recommender systems inspite of the algorithm being depicted here just collaborative filtering one. There will be minor changes at the conceptual level when implementation level changes with different vector similarity algorithms. However it is worth to discuss the complexity of these systems on the whole. The recommender systems are basically dynamic, evolving systems which should get better with every set of recommendations they predict. The main thing being captured here is the listening events i.e., the play count for every user corresponding to a song as well as the artist. The UML designs here are specifications at a high abstraction level although at a low level we need to consider the recommendation algorithm, here, the similarity score generation. The algorithms are based on various elements of the system, here, the ratings of the user to different songs. The user interface elements also play an important role in the functionality of the recommender algorithm, here, the rating element, like stars or simply a number.

The other thing to be understood is the dynamic nature of the system where with every song the user listens to, the recommendations changes, if not entirely, significantly. Also, inspite of this recommendation model not being personalized one, it still is different for every user based on his activity. The mechanism here acts as a filter to display only those songs that are relevant to the user that too in a reverse sorted order of prediction score. Every little activity of the user is to be logged and the vector values are constantly changing for song as well as artist. These things are all in the view layer of the architecture.

The logic layer deals with the implementation of the functionalities depicted in the above behavioral diagrams. Every recommendation is based on the similarity score as well as the prediction score. The persistence layer or the ‘model’ level deals with the storage of the user data, ratings, similarity matrix along with song and artist data.

The similarity score between vectors used to determine recommendation is a naive method specially for music recommendation. This naivety will cause something called as the cold start problem [2] which basically states that if there is less or no activity of new users or on new items, it may decrease the chance of them getting discovered or may distort the prediction score. The pros of this approach would be that there is inherent serendipity as the similarity is not content based. This approach also helps to learn market segments.

4. CONCLUSION:

The complex nature of the machine learning systems like the music recommender system can’t have a standardized structure because different music recommenders work in different way. This UML implementation tries to alleviate this complex nature to some extent and does so in a very abstract way. It allows to save time for developing such systems and also to incorporate object oriented features because UML supports them, greatly reducing the effort. Also, the new algorithms being devised or introduced constantly for improving the recommendations can simply be ‘plugged into’ these design structures with very little change in behavioral implementations.

The future scope of this work would be to extend this design implementation to actual implementation of live system and see the adaptability to recommender systems with varying algorithmic approaches.
5. ACKNOWLEDGEMENTS:

Extremely thankful to the reviewers for their valuable suggestions to enhance the quality of this paper.

6. REFERENCES:


AUTHOR’S BRIEF BIOGRAPHY:

**V.Sai Supreetha:** She is currently pursuing her B.Tech in the field of Information Technology at Bapatla Engineering College, Bapatla. She is a VMWare certified associate in Cloud, Data Center Virtualization and Work Force Mobility. She is a member of Student Network Team, IEEE Hyderabad Section. She is an avid Competitive Programmer and a MOOC enthusiast. She aspires to become a Data scientist by pursuing higher studies in Computer Science.

**V.S.S.K Anand:** He is currently pursuing his B.Tech in the field of Information Technology at Bapatla Engineering College, Bapatla. He is a Microsoft Student Associate and Firefox Student Ambassador. He is also a Microsoft Specialist, VMWare certified associate, WebMaker mentor for Mozilla, had organized webinars for 600+ women students as part of Microsoft Women In Tech initiative. Some of his achievements include being winner of Windows 8 App Update Challenge, Winner of Google Mapathon 2013. He aspires to get into one of the top universities for his higher studies in Computer Science.